To: WG5
Subject: US04: Non-blocking Collective Subroutines
From: Brandon Cook & Damian Rouson & Dan Bonachea
References: J3/23-174, WG5/N2334

Background
==========

In the first code example below, which is substantively similar to code presented in the Asynchronous Communication sections of the use case paper passed as 23-174, there is no reason to require that the co_sum and co_min arguments receive their collectively reduced values immediately upon completion of each collective subroutine call. Nor is there a reason to guarantee that the computed value be available before the subsequent PRINT statement.

```
real :: A=1., B=2.
call co_sum(A)
call co_min(B)
print *, “Hello world”
```

In the absence of requirements on the immediate availability of the computed values for A and B, processors would be free to allow for more overlap of communication and computation, which is a standard practice to achieve high performance and efficient use of resources in HPC. For example, a processor that relies on MPI-3 could express the independent nature of these operations using nonblocking collectives:

```
real :: A=1., B=2.
type(MPI_Request) :: R(2)
call MPI_Iallreduce(MPI_IN_PLACE, A, 1, MPI_DOUBLE, MPI_SUM, &
  MPI_COMM_WORLD, R(1))
call MPI_Iallreduce(MPI_IN_PLACE, B, 1, MPI_DOUBLE, MPI_MIN, &
  MPI_COMM_WORLD, R(2))
print *, “Hello world”
```

At a point when the collective sum of A and collective minimum of B are required, the processor could guarantee their availability with a call such as the following:

```
call MPI_Waitall(2,R,MPI_STATUSES_IGNORE)
```

where accesses to A and B are not valid until after the MPI_Waitall call.

Illustrative Proposal (NOT the final syntax)
==========

An approach to enable this feature is to introduce a new derived type with private components named completion_type, and an optional completion argument to collective subroutines similar to

```
real :: A=1., B=2.
type(completion_type) :: C
call co_sum(A, completion=C)   ! initiate non-blocking collectives
call co_min(B, completion=C)
print *, "Hello world"
...
completion wait (C, until_count=2) ! await completion of collectives
```

where A and B would only be required to attain their respective collectively reduced values after the COMPLETION WAIT statement. Additional rules about when access to participating variables (A and B above) is valid would also be needed.

The notional completion_type above would have some similarities with event_type and notify_type, but would have useful meaning without appearing in coarrays, whereas the latter two types are required to be coarrays.

Paper 23-174 was titled "Asynchronous Tasks in Fortran." The above examples, which are adapted from 23-174 while retaining the spirit of the paper, demonstrate that nonblocking collective subroutines can be implemented independently from tasks. As described here, the feature is also likely to be considerably smaller than tasks in terms of impact on the standard.

WG5 Straw Vote
==========

The following straw vote is to decide on the status of WG5 US-04 as described in WG5/N2334:

A. Keep work item WG5 US-04 unchanged and on the Accepted F202Y work list
B. Narrow the scope of F202Y work item WG5 US-04 to include only the nonblocking collective subroutines described in this paper. Defer asynchronous tasks to the work list for the next revision after 202Y
C. Defer all of work item WG5 US-04 to the work list for the next revision after 202Y